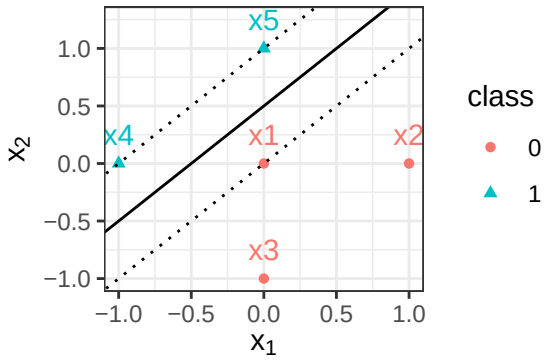
Solution 1: Logistic regression or deep learning?

a) We need to find a linear hyperplane – i.e., a line in 2D – that separates our classes. There are infinitely many such lines, but the sketch suggests that the line with intercept 0.5 and slope 1 has maximal "safety margin": it has the broadest corridor in which we could move the line without incurring a misclassification error (indicated by dotted lines). Intuitively, this leads to better generalization than any line with a smaller such margin (the mathematical reasoning is treated in support vector machines ⇝ supervised learning lecture).

Now, we need to translate this into logistic regression coefficients. Recall that the linear hyperplane for binary logistic regression is defined via

$$\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 = -\log(\tfrac{1}{c} - 1) \iff \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 = 0 \quad \text{for } c = 0.5, \tag{1}$$

where $c \in [0, 1]$ is the classification threshold.



- **Quick solution.** Logistic regression is not the focus of this exercise, so TL;DR: pick values (not unique) for which points on the line fulfill (1). For example: $\boldsymbol{\alpha}' = (-0.5, -1, 1)^\top$.
- **Detailed solution.** (1) has 3 unknown quantities – using a system of linear equations, derived from points we know our line must cross, 3 equations should suffice if there is a unique solution. It turns out that the solution for the linear hyperplane in logistic regression is *not* necessarily unique: multiplying the coefficients in (1) with a constant leaves the equality unchanged.

  In ERM, we have additional constraints that usually allow us to find an optimal parameter vector: larger $\boldsymbol{\alpha}$ driving the loss down for correctly classified observations is offset by driving it up for misclassified ones.

  However, in this special case of linear separability, where a linear classifier produces no misclassification, there is no such optimum (or rather, it is loss-optimal to push the parameter vector to infinity). We will therefore, in this highly unrealistic and simplistic example, use two points the line crosses (which is enough to define any line) and just choose some value for $\boldsymbol{\alpha}$ that fulfills (1).

  Let's pick the points $(0, 0.5)^\top$ and $(-0.5, 0)^\top$ (0's are always convenient):

$$\alpha_0 + \alpha_1 \cdot 0 + \alpha_2 \cdot 0.5 = 0 \tag{2}$$

$$\alpha_0 - \alpha_1 \cdot 0.5 + \alpha_2 \cdot 0 = 0 \tag{3}$$

  (2)-(3) yields

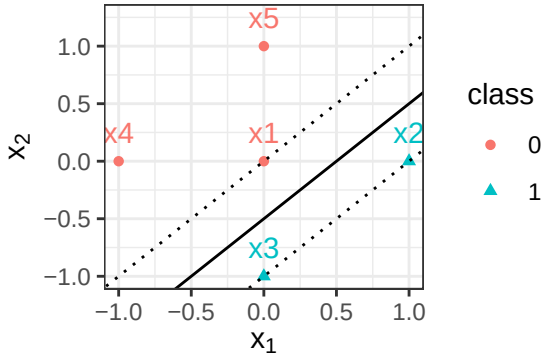
$$\alpha_1 \cdot 0.5 + \alpha_2 \cdot 0.5 = 0 \iff \alpha_1 = -\alpha_2, \tag{4}$$

  so set, e.g., $\alpha_1' = -1$ and $\alpha_2' = 1$. Plugging this back into either 2 or 3 yields $\alpha_0' = -0.5$, such that $\boldsymbol{\alpha}' = (-0.5, -1, 1)^\top$, and we get the following probabilistic scores:

i	$y^{(i)}$	$\sigma\left(\alpha'_0 + \alpha'_1 x_1^{(i)} + \alpha'_2 x_2^{(i)}\right)$
1	0	0.38
2	0	0.18
3	0	0.18
4	1	0.62
5	1	0.62

(You can easily verify that a multiple $a \cdot \boldsymbol{\alpha}'$ of $\boldsymbol{\alpha}'$ leads to the same class assignments, with probabilities closer to 0/1 if $a > 1$.)

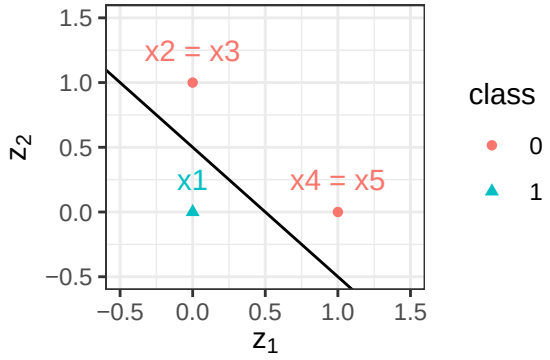
b) By the same principle as above, we can derive a possible solution $\boldsymbol{\beta}' = (-0.5, 1, -1)^\top$.



i	$y^{(i)}$	$\sigma\left(\beta'_0 + \beta'_1 x_1^{(i)} + \beta'_2 x_2^{(i)}\right)$
1	0	0.38
2	1	0.62
3	1	0.62
4	0	0.18
5	0	0.18

c)

i) Intuitively, we can see that a condition like $z_1 + z_2 > 0$ suffices to separate $\mathbf{x}^{(1)}$ from the other observations, which should remind you of the equation for a linear hyperplane.
Graphically, we can visualize our points in the new z_1, z_2 coordinates and see that this transformed dataset is indeed linearly separable:



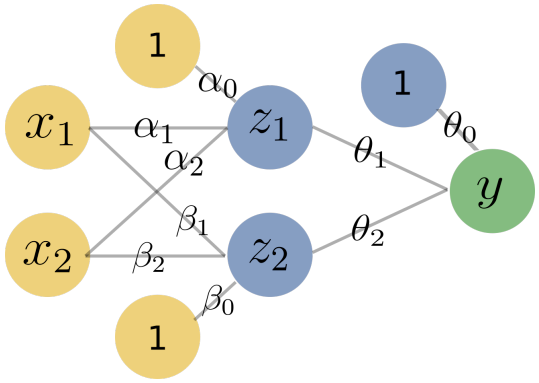
ii) Looking at z_1, we find that it serves to separate observations $\{1, 2, 3\}$ from $\{4, 5\}$. Likewise, z_2 separates $\{1, 4, 5\}$ from $\{2, 3\}$. Predicting z_1 thus corresponds to classification problem I, and z_2 to classification problem II, so we can use our previous solutions $\boldsymbol{\gamma} = \boldsymbol{\alpha}' = (-0.5, -1, 1)^\top$ and $\boldsymbol{\phi} = \boldsymbol{\beta}' = (-0.5, 1, -1)^\top$.

iii) We can simply stack the logistic regression components as follows:

$$\begin{aligned}\hat{y} &= \sigma\left(\theta_0 + \theta_1 z_1 + \theta_2 z_2\right) \\ &= \sigma\left(\theta_0 + \theta_1 \cdot \sigma\left(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2\right) + \theta_2 \cdot \sigma\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2\right)\right)\end{aligned}$$

Choosing parameters as before yields, e.g., $\boldsymbol{\theta}' = (0.5, -1, -1)^\top$. In practice, we will of course use ERM to estimate rather than hand-pick $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\theta}$.

d) We created a feedforward neural network with two input neurons, two hidden neurons (each performing one intermediate logistic regression), and one output neuron. The biases are depicted as 1-neurons:



e) The chain rule enables us to compute (first) derivatives of the empirical risk w.r.t. arbitrary parts of the computational graph by chaining known derivatives of elementary functions, thus providing a simple and scalable solution to the problem of differentiating complicated functions.

For example, we can easily derive $\frac{\partial}{\partial \alpha_1} \mathcal{R}_{\text{emp}}$, which should remind you of the update term in gradient descent. Indeed, (variants of) gradient descent is exactly what we use to train neural networks, updating all parameters according to their contribution to $\mathcal{R}_{\text{emp}}$ during the so-called *backpropagation*.